Referências Bibliográficas

[1] ATKINSON, A. C.. **Plots, transformations and regression : an introduction to graphical methods of diagnostic regression analysis**. Oxford Science Publications, Oxford, 1985.

[2] BRUNEKREEF, B.; HOLGATE, S. T.. **Air pollution and health**. Lancet, (360):1233–1242, 2002.

[3] BUJAS, A.; HASTIE, T. ; TIBSHIRANI, R.. **Linear smoothers and additive models**. The Annals of Statistics, 17(2):453–555, 1989.

[4] CAMPOS, E. L.; DE LEON, A. C. M. P. ; FERNANDES, C. A. C.. **Modelo poisson-gama para séries temporais de dados de contagem - teoria e aplicações**. 10ª ESTE - Escola de Séries Temporais e Econometria, 2003.

[5] CARNAHAN, B.; LUTHER, H. A. ; WILKES, J. O.. **Applied Numerical Methods**. John Wiley and Sons, New York, 1969.

[6] COOK, R. D.; WEISBERG, S.. **Residuals and Influence in Regression**. Monographs in Statistics and Applied probability. Chapman and Hall, New York, 1982.

[7] CRAVEN, P.; WAHBA, G.. **Smoothing noisy data with spline functions**. Numerische Mathematik, (31):377–403, 1979.

[8] DE LEON, A. P.; ANDERSON, H. R.; BLAND, J. M. ; STRACHAN, D. P.. **The effects of air pollution on daily hospital admissions for respiratory disease in london between 1987-88 and 1991-92**. Journal of Epidemiology and Community Health, 50(suppl 1):S63–S70, 1996.

[9] DURBIN, J.; KOOPMAN, S. J.. **Time Series Analysis by State Space Methods**. Oxford, New York, 2001.

[10] EUBANK, R. L.. **The hat matrix for smoothing splines**. Statistics and Probability Letters, (2):9–14, 1984.

[11] EUBANK, R. L.. **Spline Smoothing and Nonparametric Regression**. Marcel Dekker, New York, 1988.

[12] FAHRMEIR, L.; TUTZ, G.. **Multivariate Statistical Modelling Based on Generalized Linear Models**. Springer, New York, 2nd edition, 2001.

[13] FERNANDES, C. A. C.. **Non-gaussian structural time series models**. PhD thesis, London School of Economics, 1990.

[14] FRIEDMAN, J. H.; STUETZLE, W.. **Projection pursuit regression**. Journal of the American Statistical Society, 76(376):817–823, december 1971.

[15] GOUVEIA, N.; FLETCHER, T.. **Time series analysis of air pollution and mortality: effects by cause, age and socioeconomic status**. Journal of Epidemiology and Community Health, (54):750–755, 2000.

[16] GREEN, P. J.; YANDELL, B. S.. **Semi-parametric generalized linear models**. Proceedings 2nd International GLIM Conference, Lancaster, Lectures in Statistics, (32):44–55, 1985.

[17] GREEN, P.. **Linear models for field trials, smoothing and cross-validation**. Biometrika, 72(3):527–537, 1985.

[18] GREEN, P. J.; SILVERMAN, B. W.. **Nonparametric Regression and Generalized Linear Models : a roughness penalty approach**. Chapman and Hall, London, 1994.

[19] GREENLAND, S.; ROTHMAN, K. J.. **Modern Epidemiology**. Lippincot-Raven, Philadelphia, 2 edition, 1998.

[20] HARVEY, A. C.; FERNANDES, C.. **Time series models for count or qualitative observations**. Journal of Business and Economic Statistics, 7(4):407–417, 1989. with comments.

[21] HARVEY, A. C.. **Forecasting, structural time series models and the Kalman filter**. Cambridge, New York, 1990.

[22] HASTIE, T.; TIBSHIRANI, R.. **Generalized additive models**. Statistical Science, 1:297–318, 1986. with discussion.

[23] HASTIE, T. J.; TIBSHIRANI, R. J.. **Generalized Additive Models**. Chapman and Hall, London, 1990.

[24] JAMES, B. R.. **Probabilidade : um curso em nível avançado**. Projeto Euclides. Instituto de Matemática Pura e Aplicada, Rio de Janeiro, 2 edition, 1996.

[25] JUNGER, W. L.. **Imputação de dados faltando em séries temporais multivariadas via algoritmo em**. 2002.

[26] JUNGER, W. L.; CUNHA-CRUZ, J.; DA CUNHA, C. B.; PONCE DE LEON, A. ; E SILVA MENDONÇA, G. A.. **Poluição do ar e efeitos na saúde: uma abordagem dos estudos de séries temporais**. III Congresso Interamericano de Qualidade do Ar, 2003.

[27] KATSOUYANNI, K.; SCHWARTZ, J.; SPIX, C.; TOULOUMI, G.; ZMIROU, D.; ZANOBETTI, A.; WOJTYNIAK, B.; VONK, J. M.; TOBIAS, A.; PÖNKÄ, A.; MEDINA, S.; BACHÁROVÁ, L. ; ANDERSON, H. R.. **Short term effects of air pollution on health: a european approach using epidemiologic time series data: the aphea protocol**. Journal of Epidemiology and Community Health, 50(suppl 1):S12–S18, 1996.

[28] MCCULLAGH, P.; NELDER, J. A.. **Generalized Linear Models**. Chapman and Hall, 2 edition, 1989.

[29] PAPOULIS, A.. **Probability, Random Variables and Stochastic Processes**. McGraw-Hill, New York, 1965.

[30] PIERCE, D. A.; SCHAFER, D. W.. **Residuals in generalized linear models**. Journal of the American Statistical Association, 81(396):977–986, 1986.

[31] TEAM, R. D. C.. **R: A language and environment for statistical computing**. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-00-3.

[32] SCHWARTZ, J.; SPIX, C.; TOULOUMI, G.; BACHÁROVÁ, L.; BARUMAMDZADEH, T.; LE TERTRE, A.; PIEKARKSI, T.; DE LEON, A. P.; PÖNKÄ, A.; ROSSI, G.; SAEZ, M. ; SCHOUTEN, J. P.. **Methodological issues in studies of air pollution and daily counts of deaths or hospital admissions**. Journal of Epidemiology and Community Health, 50(suppl 1):S3–S11, 1996.

[33] SILVERMAN, B. W.. **Some aspects of the spline smoothing approach to non-parametric regression curve fitting**. Journal of The Royal Statistical Society, 47(1):1–52, 1985.

[34] THISTED, R. A.. **Elements of Statistical Computing : Numerical Computation**. Chapman and Hall, New York, 1988.

[35] WAHBA, G.. **Bayesian "confidence intervals"for the cross-validated smoothing spline**. Journal of the Royal Statistical Society, 45(1):133–150, 1983.

[36] WAHBA, G.. **(smoothing) splines in nonparametric regression**. Technical Report 1024, Department of Statistics - University of Wisconsin, September 2000.

[37] WOOD, S. N.. **Modelling and smoothing parameter estimation with multiple quadratic penalties**. Journal of the Royal Statistical Society, 62(2):413–428, 2000.

[38] WOOD, S. N.. **Thin plate regression spline**. Journal of the Royal Statistical Society, 65(1):95–114, 2003.

[39] YANDELL, B. S.. **Smoothing splines – a tutorial**. The Statistician, (42):317–319, 1993.

A
Código fonte em R

```
.packageName <- "pgam"
# This free software is distributed under de GNU GPL version 2 license agreement
# This source code must be copied and modified since author is cited.
# This software is free and it is provided with no warranty whatsoever
# This software can be cited as follow:
# cite the dissertation the source code is attached
#
# R code for Poisson-Gamma Additive Models (pgam)
# it is part of Washington Junger's MSc. dissertations
#
# this class of model will handle only Poisson-Gamma family for a while
# in the future it is intended to handle Negative Binomial-Beta family also.
#
# started in 11/06/2003 (dd/mm/yyyy)
# last change: (date first)
# 18/10/2003  up and running (debut) - version 0.1.0
# 19/10/2003  log(1e-7) changed to log(0.1) when y=0 (very troubling in ZIP) -
version 0.1.1
# 21/10/2003  fixed some bugs, deviance to control convergence and partial
deviance residuals in smoothing algorithm - version 0.1.2
# 22/10/2003  deviance to control convergence (not good) and deviance partial
residuals in smoothing algorithm not working - version 0.1.2
# 24/10/2003  fixed some bugs, trying partial deviance residuals once more
(works fine)
# 15/12/2003  fixed residuals degree of freedom: resdf <- n-kp-sum(sdf)-fnz
# 17/12/2003  some objects renamed
# 02/01/2004  backfitting(...) function returns the matrix of partial residuals
now - version 0.1.3
# 03/01/2004  resource consuming envelope plot implemented now, some bugs fixed
- version 0.1.4
# 12/01/2004  function call is returned now - version 0.1.5
# 05/02/2004  minor changes in envelope routine - version 0.1.6
#
#
# To do list:
# -----------
# * properly estimate s.e. of the last seasonal factor (it needs a function like
par2psi())
# * implement the analytical method to estimate information matrix
# * some built-in plots
# * summary output
# * correct estimation of spectrum
# * using standardized deviance residuals in envelope can halt function --> to
be fixed
#
#
```

```
# functions begin here ----------------------

pgam.parser <- function(formula,parent.level=1)
# reads the model formula and splits it in two new formulae: one of parametric
# terms and another for smoothed terms.
{
formterms <- terms.formula(formula,specials=c("g","f")) # s'ed terms is supposed
to be smoothed (g) and factorized (f)
modterms <- attr(formterms,"term.labels") # assigns labels of model terms
nterms <- length(modterms) # number of terms in model
fformula <- NULL
pformula <- NULL # parametric formula
sformula <- NULL  # smoothing formula
oterm <- NULL  # offset term
sdf <- NULL  # smoothing df vector
sfx <- NULL  # smoothing method fixed vector
findex <- NULL
pindex <- NULL
sindex <- NULL
oindex <- NULL
fnames <- NULL
fdata <- NULL
fperiod <- NULL

response <- attr(formterms,"response")
if (!is.null(response))
response <-
as.formula(paste("~",as.character(attr(formterms,"variables")[2]),sep=""))
else
stop("Error: Response variable not specified.")

if (!is.null(attr(formterms,"specials")$f))
findex <- attr(formterms,"specials")$f # indeces of terms to be factorized in
modterms array
if (!is.null(attr(formterms,"specials")$g))
sindex <- attr(formterms,"specials")$g # indeces of terms to be smoothed in
modterms array
if (!is.null(attr(formterms,"offset")))
oindex <- attr(formterms,"offset") # offset location

if (nterms)
for (k in 2:(nterms+1))
if (!(k %in% sindex) && !(k %in% oindex) && !(k %in% findex))
     pindex <- c(pindex,k)  # gets parametric terms indeces

fterms <- length(findex) # number of terms in factorized formula
pterms <- length(pindex) # number of terms in parametric formula
sterms <- length(sindex) # number of terms in smooth formula

if (fterms)
{
    for (k in 1:fterms)  # seasonal factors formula - one ocurrence only !!!!
must change!!!
     {
     fed <-
eval(parse(text=as.character(attr(formterms,"variables")[findex[k]+1]))) # f
extraction
        for (d in 1:(fed$fperiod-1))
         fformula <-
paste(fformula,fed$factornames[d],sep=ifelse(is.null(fformula),"~","+"))
     fnames <- c(fnames,fed$factornames)
     fdata <- c(fdata,fed$factordata)
        fperiod <- c(fperiod,fed$fperiod)
}
fformula <- as.formula(fformula)
    fdata <- as.matrix(as.data.frame(fdata))
    }

if (pterms)
{
for (k in 1:pterms)  # parametric formula
pformula <-
paste(pformula,as.character(attr(formterms,"variables")[pindex[k]+1]),sep=ifelse
(is.null(pformula),"~","+"))
pformula <- as.formula(pformula)
    }

if (sterms)
{
    for (k in 1:sterms)  # non-parametric formula
     {
     sed <-
eval(parse(text=as.character(attr(formterms,"variables")[sindex[k]+1]))) # g
extraction
sformula <- paste(sformula,sed$var,sep=ifelse(is.null(sformula),"~","+"))
     sdf <- c(sdf,sed$df)
     sfx <- c(sfx,sed$fx)
}
sformula <- as.formula(sformula)
    }

if (!is.null(oindex))
oterm <-
as.formula(paste("~",eval(parse(text=as.character(attr(formterms,"variables")[oi
ndex+1])))$var,sep=""))
# the poisson-gamma model does not have intercept. i will keep this for future
use as deterministic drift.
if (attr(formterms,"intercept") == 0)
    drift <- FALSE
else
drift <- TRUE

retval <-
list(response=response,pformula=pformula,pterms=pterms,sformula=sformula,sdf=sdf
,sfx=sfx,sterms=sterms,fterms=fterms,fformula=fformula,fnames=fnames,fdata=fdata
,fperiod=fperiod,offset=oterm,drift=drift,fullformula=formula)
class(retval) <- "pgam.split.formula"
return(retval)
}


pgam.filter <- function(w,y,eta)
# runs appropriate recursions for omega estimate
{
n <- length(y)
att1 <- double(n)
btt1 <- double(n)
at <- double(n+1)
```

```
bt <- double(n+1)
at[1] <- 0.0
bt[1] <- 0.0

for (t in 1:n)
{
att1[t] <- w*at[t]
    at[t+1] <- w*at[t]+y[t]
    # explanatory variables model is considered if eta != NULL
    if (!is.null(eta))
     {
        btt1[t] <- w*bt[t]*exp(-eta[t])
     bt[t+1] <- w*bt[t]+exp(eta[t])
     }
    else
      {
        btt1[t] <- w*bt[t]
     bt[t+1] <- w*bt[t]+1
    }
}
retval <- list(att1=att1,btt1=btt1,at=at[2:(n+1)],bt=bt[2:(n+1)])
return(retval)
}


pgam.likelihood <-
function(par,y,x,offset,fperiod,trace,digits,env=parent.frame())
# likelihood function to be maximized
{
# splitting parameter vector into omega and beta
psi <- pgam.par2psi(par,fperiod)
fnz <- fnz(y)
n <- length(y)
lvalue <- 0.0

if (trace)
{
cat("psi: ");cat(round(c(psi$w,psi$beta),digits));cat("\n")
}

if (!is.null(psi$beta))
eta <- x%*%psi$beta+offset  # parametric piece of predictor
else
eta <- NULL

filtered <- pgam.filter(psi$w,y,eta)
att1 <- filtered$att1
btt1 <- filtered$btt1

for (t in (fnz+1):n) # on mle, sumation starts at the first non-zeron obs
    lvalue <-
lvalue+lgamma(att1[t]+y[t])-lfact(y[t])-lgamma(att1[t])+att1[t]*log(btt1[t])-(at
t1[t]+y[t])*log(1+btt1[t])

loglik <- list(value=lvalue,eta=eta,att1=att1,btt1=btt1)
assign("loglik",loglik,env=env)

return(lvalue)
}
```

```
pgam.fit <- function(w,y,etap,etas=NULL)
# estimates of y_hat_t|t-1
{
if (is.null(etas))
{
    smopredictor <- NULL
    etas <- 0
    }
else
smopredictor <- 1

if (is.null(etap) && is.null(smopredictor))
eta <- NULL
else
eta <- etap+etas

filtered <- pgam.filter(w,y,eta)
at <- filtered$at
bt <- filtered$bt
att1 <- filtered$att1
btt1 <- filtered$btt1

n <- length(y)
yhat <- double(n)
vyhat <- double(n)
deviance <- double(n)
pearson <- double(n)
hat <- double(n)
fnz <- fnz(y)
level <- NULL
yhats <- NULL

for (t in fnz:n)
{
    # prediction
    yhat[t] <- att1[t]/btt1[t]
    vyhat[t] <- att1[t]*(1+btt1[t])/btt1[t]^2
    # getting deviance component
if (!(y[t] == 0))
     deviance[t] <-
2*(att1[t]*log(att1[t]/(y[t]*btt1[t]))-(att1[t]+y[t])*log((y[t]+att1[t])/((1+btt
1[t])*y[t])))
else
     deviance[t] <- 2*(att1[t]*log((1+btt1[t])/btt1[t]))
    # getting generalized pearson statistics component
pearson[t] <- ((y[t]*btt1[t]-att1[t])^2)/(att1[t]*(1+btt1[t]))
}

if (is.null(eta))
eta <- rep(0,n)
for (t in 1:(n-1))
{
# pseudo hat matrix
    hat[t] <- w*exp(eta[t+1])/sum(w^(0:(t-1))*exp(eta[(t-0):1]),na.rm=T)
}

# an attempt of extraction of components
```

```
if (!is.null(smopredictor))
{
    # semiparametric model
    plevel <- yhat*exp(-etap)
level <- plevel*exp(-etas)
yhats <- plevel/level
    }
else if (!is.null(etap))
{
    # full parametric model
    level <- yhat*exp(-etap)
    yhats <- NULL
    }
else if (is.null(etap))
level <- yhat

# residuals are to be extracted elsewhere

# cleaning the house
yhat[1:fnz] <- NA
vyhat[1:fnz] <- NA
deviance[1:fnz] <- NA
pearson[1:fnz] <- NA

retval <-
list(yhat=yhat,vyhat=vyhat,deviance=deviance,pearson=pearson,hat=hat,level=level
,yhats=yhats)
return(retval)
}


pgam.psi2par <- function(w,beta,fperiod)
# puts hyperparameters into optmization form
{
alpha <- log(w/(1-w)) # transformation to ensure constraints on w
fp <- length(fperiod)
newbeta <- NULL

if (!is.null(beta))
{
fp <- length(fperiod)
    bk <- 1

    if (!is.null(fperiod))
     {
        for (k in 1:fp)
     {
        newbetak <- beta[bk:(fperiod[k]-1)]
         bk <- length(newbetak)+2  # skip the last seasonal factor ex.:
saturday, december etc
         newbeta <- c(newbeta,newbetak)
         }
     if (length(beta) > sum(fperiod))
     newbeta <- c(newbeta,beta[bk:length(beta)])
}
else
     newbeta <- beta
    }
par <- c(alpha,newbeta) # parameter vector to be optimized
```

```
return(par)
}


pgam.par2psi <- function(par,fperiod)
# puts parameters optmization form back into beta
{
alpha <- par[1]
w <- exp(alpha)/(1+exp(alpha))
beta <- NULL

if (length(par) > 1)
{
newbeta <- par[2:length(par)]
fp <- length(fperiod)
bk <- 1

    if (!is.null(fperiod))
     {
for (k in 1:fp)
     {
         newbetak <- newbeta[bk:(bk+fperiod[k]-2)]  # get pieces of beta from
par
         bk <- length(newbetak)+1
         beta <- c(beta,newbetak,-sum(newbetak))
         }
    if (length(newbeta) >= sum(fperiod))  # something odd is going on about
here!
    beta <- c(beta,newbeta[bk:length(newbeta)])
}
else
     beta <- newbeta
    }
retval <- list(w=w,beta=beta)
return(retval)
}


pgam.hes2se <- function(hes,fperiod)
# puts parameters hessian matrix in the form of beta s.e.
{
alpha <- par[1]
w <- exp(alpha)/(1+exp(alpha))
beta <- NULL

if (length(par) > 1)
{
newbeta <- par[2:length(par)]
fp <- length(fperiod)
bk <- 1

    if (!is.null(fperiod))
for (k in 1:fp)
     {
         newbetak <- newbeta[bk:(bk+fperiod[k]-2)]  # get pieces of beta from
par
         bk <- length(newbetak)+1
         beta <- c(beta,newbetak,-sum(newbetak))
         }
    beta <- c(beta,newbeta[bk:length(newbeta)])
    }
retval <- list(w=w,beta=beta)
return(retval)
}


pgam <-
function(formula,dataset,omega=0.8,beta=0.01,offset=NULL,digits=5,maxit=1e2,eps=
1e-6,control=list(trace=10,maxit=1e2,abstol=1e-4,reltol=1e-3,factr=1e7,pgtol=0),
optim.method="BFGS",trace=F,partial.resid="response",smoother="spline",bkf.eps=1
e-3,bkf.maxit=1e2,numerical.se=T)
# estimates Poisson-Gamma Additive Models
{
st <- proc.time()
called <- match.call()
pgam.env <- new.env(FALSE,NULL)  # new environment defined for pgam

if (is.null(formula))
stop("Error: Model formula is not specified.")
if (is.null(dataset))
stop("Error: Model dataset is not specified.")

parsed <- pgam.parser(formula)  # parse the formula and get components

# building datasets and setting the model structure (no explanatory variables,
seasonal factors, linear, additive, offset, drift)
response <- framebuilder(parsed$response,dataset)$frame
yname <- names(response)
y <- response[[1]]
n <- length(y)  # get number of obs
if (sum((y-as.integer(y))>0))
{
    y <- as.integer(y) # non-integer values are truncated
    cat("Note: Some values must have been truncated in order to ensure
compliance with Poisson specification.\n")
    }
else
    y <- as.integer(y) # to avoid type conflict with integer functions

if (sum(y < 0) > 0) # checking for negative values. if detected, program halts
stop("Error: Negative observations detected. The series does not comply with
Poisson specification.")
fnz <- fnz(y)
undef <- rep(0,fnz)

etap <- NULL
px <- NULL
kx <- 0
pnames <- NULL
if (parsed$pterms)
{
    pform <- parsed$pformula
px <- framebuilder(pform,dataset)$frame
    pnames <- names(px)
    px <- as.matrix(px)
    kx <- dim(px)[2]  # number of non-seasonal factor explanatory variables
    }
```

```
etas <- NULL
bkf <- NULL
sx <- NULL
sdf <- NULL
sfx <- NULL
ks <- 0
snames <- NULL
if (parsed$sterms)
{
    sform <- parsed$sformula
sx <- framebuilder(sform,dataset)$frame
    snames <- names(sx)
    sx <- as.matrix(sx[(fnz+1):n,])
    sdf <- parsed$sdf
    sfx <- parsed$sfx
ks <- dim(sx)[2] # number of non-parametric explanatory variables
}

fx <- NULL
fp <- 0
kf <- 0
fperiod <- NULL
fnames <- NULL
if (parsed$fterms)
{
    fform <- parsed$fformula
fx <- parsed$fdata
    fnames <- parsed$fnames
    fperiod <- parsed$fperiod
    fp <- length(fperiod)  # number of seasonal factors terms
    kf <- sum(fperiod) # total count of seasonal factors
    }

oterm <- offset
offset <- rep(0,n)
if (!is.null(parsed$offset))
{
    offset <- framebuilder(parsed$offset,dataset)$frame
    }

if (parsed$drift)
{
    # to be worked out
    }

px <- cbind(fx,px)
kp <- kx+kf # number of explanatory variables

if (length(beta) == 1)
beta <- rep(beta,kp) # expansion of initial value of beta vector (assuming all
the same)
if (kp == 0)
beta <- NULL
if (is.null(px) && !is.null(sx))
    stop("Error: This application still not fit full non-linear predictor
models.")

assign("loglik",NULL,env=pgam.env)
```

```
# mle estimation
par <- pgam.psi2par(omega,beta,fperiod)

optimized <-
optim(par,pgam.likelihood,y=y,x=px,offset=offset,fperiod=fperiod,trace=trace,dig
its=digits,env=pgam.env,method=optim.method,hessian=F,control=c(control,REPORT=c
ontrol$trace,fnscale=-1,))
newoffset <- offset  # case of full parametric model

# smoothing
k <- 0  # if k=0 at the end o function, this is a full parametric model
norm <-0  # same as k
if (!is.null(sx))
{
    norm <- 1e35  # large number intialization of norm
    ynz <- y+0.1*(y==0)  # replaces zero counts by a small value (dangerous!!!)
    # oldetas <- 0
    loglik0 <- 0
    # deviance0 <- 0
    k <- 0
    for (k in 1:maxit)
      {
        # getting useful information
psi <- pgam.par2psi(optimized$par,fperiod)
        etap <- px%*%psi$beta  # parametric piece of predictor
predicted <- pgam.fit(psi$w,y,etap+offset)

        if (partial.resid == "response")
         presid <- (log(ynz)-log(predicted$yhat))[(fnz+1):n]
        else if (partial.resid == "deviance")
         presid <- (sign(y-predicted$yhat)*sqrt(predicted$deviance))[(fnz+1):n]
        loglik1 <- (-1)*optimized$value
        # deviance1 <- sum(predicted$deviance,na.rm=T)
        # backfitting smoothing
        bkf <-
backfitting(presid,sx,sdf,sfx,smoother=smoother,eps=bkf.eps,maxit=bkf.maxit,info
=F)
        etas <- c(undef,bkf$sumfx)
       newoffset <- offset+etas  # preserves original offset in full eta
        # norm <- lpnorm(etas,oldetas,p=1)/lpnorm(oldetas,p=1)
        norm <- abs((loglik1-loglik0)/loglik0)
        # norm <- abs((deviance1-deviance0)/deviance0)
cat(paste("iter:",k," | ","norm:",round(norm,digits),"\n"))

        if (!(norm < eps))
            {
     if (k > maxit)
             {
             cat("\nWarning: No convergence after last iteration of the
estimation algorithm.\n")
                break
             }
            # oldetas <- etas
            loglik0 <- loglik1
            # deviance0 <- deviance1
            # parametric fitting
            optimized <-
optim(optimized$par,pgam.likelihood,y=y,x=px,offset=newoffset,fperiod=fperiod,tr
ace=trace,digits=digits,env=pgam.env,method=optim.method,hessian=F,control=c(con
trol,REPORT=control$trace,fnscale=-1,))
            }
         else
  {
cat("\nSemiparametric model estimation algorithm has converged.\n")
            break
}
        }
}

# last running in order to get evaluated functions and hessian matrix
cat("\nFinal run: Getting estimated parameters, functions and numerical hessian
matrix...\n")
optimized <-
optim(optimized$par,pgam.likelihood,y=y,x=px,offset=newoffset,fperiod=fperiod,tr
ace=trace,digits=digits,env=pgam.env,method=optim.method,hessian=numerical.se,co
ntrol=c(control,REPORT=control$trace,fnscale=-1,))
psi <- pgam.par2psi(optimized$par,fperiod)
if (!is.null(sx))
{
    etap <- px%*%psi$beta  # parametric piece of predictor
    predicted <- pgam.fit(psi$w,y,etap+offset)
    if (partial.resid == "response")
    presid <- (log(ynz)-log(predicted$yhat))[(fnz+1):n]
    else if (partial.resid == "deviance")
     presid <- (sign(y-predicted$yhat)*sqrt(predicted$deviance))[(fnz+1):n]
    # backfitting smoothing
    bkf <-
backfitting(presid,sx,sdf,sfx,smoother=smoother,eps=bkf.eps,maxit=bkf.maxit,info
=T)
    # restoring original size of elements in bkf and sx
bkf$sumfx <- c(undef,bkf$sumfx)
    bkf$smox <- rbind(matrix(NA,fnz,ks),bkf$smox)
dimnames(bkf$smox) <- list(NULL,snames)
    bkf$pres <- rbind(matrix(NA,fnz,ks),bkf$pres)
dimnames(bkf$pres) <- list(NULL,snames)
sx <- rbind(matrix(NA,fnz,ks),sx)
    }

# getting useful information
omega <- psi$w
names(omega) <- c("(Discount)")

beta <- psi$beta
if (!is.null(psi$beta))
{
    etap <- px%*%beta+offset  # parametric piece of predictor
    if (!is.null(sx))
     {
     etas <- bkf$sumfx
fitted <- pgam.fit(omega,y,etap,etas)
        }
     else
      {
      etas <- rep(0,n)
        fitted <- pgam.fit(omega,y,etap+etas)
        }
    }
else
```

```
fitted <- pgam.fit(omega,y,NULL)
se.omega <- NA
se.beta <- NA
if (numerical.se == T)
{
    names(beta) <- c(fnames,pnames)
    # temporary solution for s.e.
covar <- try(solve(-optimized$hessian))
se <- pgam.par2psi(sqrt(diag(covar)),fperiod)
    alpha <- par[1]
    se.omega <- se$w*(exp(alpha)/(1+exp(alpha))^2)  # correcting se using delta
rule James, B. (1996), Probabilidade:..., p.253
    names(se.omega) <- c("(Discount)")
    se.beta <- se$beta
    se.beta[fperiod[1]] <- NA  # temporary solution to avoid misunderstandings
in output
names(se.beta) <- c(fnames,pnames)
    }

iterations <- optimized$counts
cat("Counts (fn | gr): ");cat(iterations);cat("\n")

if (!is.null(optimized$convergence))
if (optimized$convergence == 0)
{
        convergence <- "converged"
        }
else
convergence <- "not converged"

loglik.value <- (-1)*optimized$value
loglik <- get("loglik",env=pgam.env)
eta <- etap+etas
att1 <- loglik$att1
btt1 <- loglik$btt1

# corrected estimated residuals degrees of freedom
if (!is.null(bkf))
edf <- kp+fnz+sum(bkf$edf)
else
edf <- kp+fnz
resdf <- n-edf

# scale parameter based on generalized Pearson statitics
scale <- sum(fitted$pearson,na.rm=T)/resdf

dataset <- deparse(substitute(dataset))

et <- proc.time()
cat(paste("\nEstimation process took", elapsedtime(st,et)),"(hh:mm:ss)\n")

retval <-
list(call=called,formula=formula,dataset=dataset,omega=omega,se.omega=se.omega,b
eta=beta,se.beta=se.beta,att1=att1,btt1=btt1,loglik=loglik.value,convergence=con
vergence,optim.method=optim.method,yhat=fitted$yhat,vyhat=fitted$vyhat,deviance=
fitted$deviance,pearson=fitted$pearson,hat=fitted$hat,level=fitted$level,yhats=f
itted$yhats,y=y,px=px,sx=sx,offset=oterm,eta=eta,etas=etas,bkf=bkf,alg.k=k,alg.n
orm=norm,edf=edf,resdf=resdf,n=n,tau=fnz,scale=scale)
class(retval) <- "pgam"
```

```
return(retval)
}


residuals.pgam <- function(object,...,type="deviance")
# method for residuals extraction of a pgam model object
{
# the following is bugging me!!!
#if (!exists(deparse(substitute(model))))
# stop("Error: Model supplied does not exist in current database.")

# adopting GLM notation for residuals extraction
y <- object$y
mu <- object$yhat
v.mu <- object$vyhat
d <- object$deviance
h <- object$hat
att1 <- object$att1
btt1 <- object$btt1
phi <- object$scale

raw <- y-mu  # getting raw residuals

if (type == "response")
resid <- raw
else if (type == "pearson")
resid <- raw/sqrt(v.mu)
else if (type == "deviance")
resid <- sign(raw)*sqrt(d)
else if (type == "std_deviance")
resid <- sign(raw)*sqrt(d/(1-h))
else if (type == "std_scl_deviance")
resid <- sign(raw)*sqrt(d/(phi*(1-h)))
else if (type == "adj_deviance")
{
    skew <- (2-btt1)/sqrt(att1*(1-btt1))  # skewness coeficient of negative
binomial distribution
resid <- sign(raw)*sqrt(d)*skew
    }
else
stop(paste("Error: Residuals of type",type,"is not implmented yet."))

retval <- resid
return(retval)
}


predict.pgam <- function(object,...)
# method for prediction for new values
{
cat("\nNot implemented yet!\nIt will be soon enough!\n")
retval <- NULL
return(retval)
}


fitted.pgam <- function(object,...)
# method for fitted extraction only - for compatibility
{
```

```
retval <- object$yhat
return(retval)
}


coef.pgam <- function(object,...)
# method for parametric coefficients extraction. although omega is not a coef,
it is output at first slot.
{
retval <- c(object$omega,object$beta)
return(retval)
}


logLik.pgam <- function(object,...)
# logLik method for extraction of loglik from a pgam object
{
retval <- object$loglik
return(retval)
}


deviance.pgam <- function(object,...)
# deviance method for extraction of deviance from a pgam object
{
retval <- sum(object$deviance,na.rm=T)
return(retval)
}


AIC.pgam <- function(object,...,k=2)
# method for AIC estimation of a pgam model object
{
# gathering necessary quantities
nprime <- object$n-object$tau
deviance <- sum(object$deviance,na.rm=T)
edf <- object$edf
phi <- object$scale

retval <- (1/nprime)*(deviance+k*edf*phi)
return(retval)
}


summary.pgam <- function(object,...)
# method for output summary of pgam model object
{
# general stuff
call <- object$call
formula <- object$formula
convergence <- object$convergence
optim.method <- object$optim.method
n <- object$n
tau <- object$tau
alg.k <- object$alg.k
alg.norm <- object$alg.norm
# goodness-of-fit statistics
deviance <- sum(object$deviance,na.rm=T)
pearson <- sum(object$pearson,na.rm=T)
```

```
scale <- object$scale
edf <- object$edf
res.edf <- object$resdf
# maximum likelihood parameters and hypothesis testing
loglik <- object$loglik
coeff <- c(object$omega,object$beta)
se.coeff <- c(object$se.omega,object$se.beta)
t.coeff <- coeff/se.coeff
pt.coeff <- 2*(1-pt(abs(t.coeff),df=res.edf))
if (!is.null(object$bkf))
{
# nonparametric stuff
vars.smo <- dimnames(object$bkf$smox)[[2]]
edf.smo <- object$bkf$edf
# approximate F-tests must be inserted at this point (soon!)
chi.smo <- as.double(rep(NA,length(vars.smo)))
pchi.smo <- as.double(rep(NA,length(vars.smo)))
}
else
{
vars.smo <- NULL
edf.smo <- NULL
chi.smo <- NULL
pchi.smo <- NULL
}

retval <-
list(call=call,formula=formula,convergence=convergence,optim.method=optim.method
,n=n,tau=tau,alg.k=alg.k,alg.norm=alg.norm,deviance=deviance,pearson=pearson,sca
le=scale,edf=edf,res.edf=res.edf,loglik=loglik,coeff=coeff,se.coeff=se.coeff,t.c
oeff=t.coeff,pt.coeff=pt.coeff,vars.smo=vars.smo,edf.smo=edf.smo,chi.smo=chi.smo
,pchi.smo=pchi.smo)
class(retval) <- "summary.pgam"
return(retval)
}


print.summary.pgam <- function(object,digits=getOption("digits"),...)
# method for summary printing
{
cat("Function call:\n")
print(object$call)
cat("\nModel formula:\n")
print(object$formula)
cat("\nParametric coefficients:\n")
width <- max(nchar(names(object$coeff)))+2
cat(rep(" ",width),"     Estimate     std. err.     t ratio
Pr(>|t|)\n",sep="")
for (i in 1:length(object$coeff))
    cat(formatC(names(object$coeff)[i],width=width),"
",formatC(object$coeff[i],width=digits+6,digits=digits),"
",formatC(object$se.coeff[i],width=digits+6,digits=digits),"
",formatC(object$t.coeff[i],width=digits+6,digits=digits),"
",format.pval(object$pt.coeff[i]),"\n",sep="")
# nonparametric partition of the model
if (!is.null(object$vars.smo))
{
cat("\nApproximate significance of smooth terms:\n")
    width <- max(nchar(object$vars.smo))+2
    cat(rep(" ",width),"           edf       chi.sq     p-value\n",sep="")
    for (i in 1:length(object$vars.smo))
cat(formatC(object$vars.smo[i],width=width),"
",formatC(object$edf.smo[i],width=digits+6,digits=digits),"
",formatC(object$chi.smo[i],width=digits+6,digits=digits),"
",format.pval(object$pchi.smo[i]),"\n",sep="")
}
cat("\nLog-likelihood value is ",round(object$loglik,digits)," after
",object$optim.method," has ",object$convergence,".\n",sep="")
if (object$alg.k > 0)
cat("\nEstimation process of semiparametric model stopped after
",object$alg.k," iterations at ",round(object$alg.norm),".\n",sep="")
else
cat("\nFull parametric model estimated.\n")
cat("\nResidual deviance is ",round(object$deviance,digits)," on
",object$res.edf," estimated degrees of freedom.\n",sep="")
cat("\nApproximate dispersion parameter equals ",round(object$scale,digits),"
based on generalized Pearson statistics
",round(object$pearson,digits),".\n",sep="")
cat("\nDiffuse initialization wasted the ",object$tau," first
observation(s).\n",sep="")
}


plot.pgam <- function(object,rug=T,se=T,at.once=F,...)
# method for smooth terms plotting
{
#gathering some useful information
n <- object$n
y.level <- object$level
x.level <- seq(1:n)
edf <- object$bkf$edf

# plotting level
plot(x.level,y.level,type="l",xlab="time",ylab="local level",...)
if (rug)s
rug(x.level)

# plotting smoothed covariates in predictor scale
if (!is.null(edf))
{
s <- length(edf)
vars.smo <- dimnames(object$bkf$smox)[[2]]
x.g <- object$sx
y.g <- object$bkf$smox
# setting same scale to smoothed covariates plots - must be updated when
plotting se band!!!
y.g.lim <- c(min(y.g,na.rm=T),max(y.g,na.rm=T))
#setting labels
y.g.lab <- paste("g(",vars.smo,",",edf,")",sep="")
for (i in 1:s)
{
if (at.once)
X11()
else
if (interactive())
readline("Press ENTER for next page....")
# must be in appropriate order
x <- x.g[order(x.g[,i]),]
y <- y.g[order(x.g[,i]),]
plot(x[,i],y[,i],type="l",ylim=y.g.lim,xlab=vars.smo[i],ylab=y.g.lab[i],...)
rug(x.g[,i])
}
}
else
cat("\nFull parametric model. Nothing left for plot.pgam() function to do.\n")
}


f <- function(factorvar)
# builds factor data matrix
{
factorname <- deparse(substitute(factorvar))
factorvar <- as.matrix(factorvar)
n <- length(factorvar)
m <- max(factorvar)
factordata <- as.data.frame(matrix(NA,n,m))
datanames <- NULL
for (k in 1:m)
{
    factordata[k] <- 1*(factorvar == k)
    datanames <- c(datanames,paste(factorname,k,sep="."))
    }
names(factordata) <- datanames

retval <- list(factordata=factordata,factornames=datanames,fperiod=m)
class(retval) <- "pgam.factor.info"
return(retval)
}


g <- function(var,df,fx=T)
# extracts spline information from the formula term g(var,df,fx)
{
varname <- deparse(substitute(var))
retval <- list(var=varname,df=df,fx=fx)
class(retval) <- "pgam.spline.info"
return(retval)
}


offset <- function(var)
# extracts offset information from the formula term offset(var)
{
varname <- deparse(substitute(var))
retval <- list(var=varname)
class(retval) <- "pgam.offset.info"
return(retval)
}


lfact <- function(x)
# calculates the log-factorial of x, i. e., log(x!)
{
if (sum((x-as.integer(x))!=0))
return(-1)  # non-integer error code
else
retval <- lgamma(x+1)  # log(gamma(k))=log((k-1)!)
```

```
return(retval)
}


fnz <- function(y)
# returns first non-zero observation index (base 1)
{
for (t in 1:length(y))
if (y[t] > 0)
     return(t)
}


framebuilder <- function(formula,dataset)
# builds a data frame from the dataset given the formula
{
if (!is.null(formula))
frame <- model.frame(formula,dataset)
else
frame <- NULL
retval <- list(frame=frame)
class(retval) <- "pgam.data.frame"
return(retval)
}


varbuilder <- function(varname,dataset)
# builds a var vector from the dataset given the varname
{
if (!is.null(varname))
var <- eval(parse(text=varname),envir=dataset)
else
var <- NULL
retval <- list(var=var,varname=varname)
class(retval) <- "pgam.data.var"
return(retval)
}


backfitting <-
function(y,x,df,fx,smoother="spline",w=rep(1,length(y)),eps=1e-3,maxit=1e2,info=
T)
# ajust the non-parametric piece of predictor
{
# getting useful information
n <- dim(x)[1] # number of observations
p <- dim(x)[2] # number of variables to be smoothed

# initialization
smox <- matrix(0,n,p)  # create storage space for smoothed variables
pres <- matrix(0,n,p)  # create storage space for partial residuals variables
lev <- matrix(0,n,p)  # create storage space for smoother leverage
edf <- double(p)  # create storage space for edf
norm <- 1e35 # large value
smooth <- rep(0,n)
m <- 0

#backfitting
```

```
while ((norm >= eps) && (m <= maxit))
{
m <- m+1
    oldsmooth <- smooth
    smooth <- rep(0,n)
    for (j in 1:p)
{
sumfx <- rep(0,n)
        for (k in 1:p)
            smox[,k] <-
(k!=j)*pgam.smooth(y,x[,k],df[k],fx[k],smoother=smoother,w=w)$fitted
        for (k in 1:p)
            sumfx <- sumfx+(k!=j)*smox[,k]
        pres[,j] <- y-sumfx
        smoothed <-
pgam.smooth(pres[,j],x[,j],df[j],fx[j],smoother=smoother,w=w)
smox[,j] <- smoothed$fitted
# lev[,j] <- smoothed$lev
# edf[j] <- smoothed$df
        smooth <- smooth+smox[,j]
        }

norm <- lpnorm(smooth,oldsmooth,p=2)/lpnorm(oldsmooth,p=2)
}

sumfx <- apply(smox,1,sum)  # sum of fx
if (!info)
{
# short returning list - intended to be fast!
retval <- list(sumfx=sumfx)
}
else
{
# complete returning list - slower!

# in future, when cross-validation be implemented, this will handle the
estimated degrees of freedom according to the approximation of Hastie &
Tibshirani (1990) given by df_j=trH_j(lambda_j)-1. as df is supposed to be fixed
(yet), this will only penalize de degrees of freedom of the spline g_j
edf <- df-1
retval <- list(sumfx=sumfx,smox=smox,pres=pres,lev=lev,edf=edf)
}
class(retval) <- "backfitting"
return(retval)
}


pgam.smooth <- function(y,x,df,fx,smoother="spline",w=rep(1,length(y)))
# smooths y against x wiht ds degrees of smoothness
{
if (smoother=="spline")
{
smoothed <- smooth.spline(x=x,y=y,w=w,df=df)
fitted <- predict(smoothed,x)$y
lev <- smoothed$lev
df <- smoothed$df
}
#else if (smoother=="loess")
# {
```

```
# tempds <- as.data.frame(cbind(y,x)) # temporary dataset
# names(tempds) <- c("y","x")
# fitted <- loess(as.formula("y~x"),tempds,weights=w,span=1/df)$fitted
#   retval <- list(fitted=fitted)
# }
else
stop(paste("Error: Smoother",smoother,"not implemented yet."))

retval <- list(fitted=fitted,lev=lev,df=df)
return(retval)
}


elapsedtime <- function(st,et)
# Computes the elapsed time between st (start time) and et (end time)
{
time <- et[3]-st[3] # gets time from the third position of the vectors
h <- trunc(time/3600)
if (h<10)
hs <- paste("0",h,sep="",collapse=" ")
else
hs <- h
time <- time-h*3600
min <- trunc(time/60)
if (min<10)
mins <- paste("0",min,sep="",collapse=" ")
else
mins <- min
time <- time-min*60
sec <- trunc(time)
if (sec<10)
secs <- paste("0",sec,sep="",collapse=" ")
else
secs <- sec

retval <- paste(hs,":",mins,":",secs,sep="",collapse=" ")
return(retval)
}


link <- function(x,link="log",inv=F)
# apllies the link function
{
if (link=="log")
{
if (!inv)
linked <- log(x)
else
linked <- exp(x)
}
else
stop(paste("Error: Link funtion",link,"not implemented yet."))
return(linked)
}


lpnorm <- function(seq1,seq2=0,p=0)
# returns the Lp-norm. If p=0 then Infinity norm is returned
{
```

```
if (p == 0)
norm <- max(abs(seq1-seq2),na.rm=T)
if (p == 1)
norm <- sum(abs(seq1-seq2),na.rm=T)
if (p >= 2)
norm <- sum((seq1-seq2)^p,na.rm=T)
if (p > 2)
warning("Lp-norm where p is greater than 2 is quite unusual.")

return(norm)
}


intensity <- function(x,n,y)
# spectral analysis of series y
{
t <- seq(1:n)

sp <- ((sum(y*cos(x*t)))^2+(sum(y*cos(x*t)))^2)/n
return(sp)
}


periodogram <- function(x,rows=12,title=NULL,...)
# creates and plots periodogram of series x
{
# initialization
if (is.null(title))
title <- paste("Periodogram of series",deparse(substitute(x)))
x <- na.exclude(x)
n <- length(x)
IOmega <- NULL
i <- seq(1:trunc(n/2-1))
t <- seq(1:n)
omega <- (2*pi*i)/n

IOmega <- sapply(i,function(x){intensity(omega[x], n=n, y=x)})
period <- (2*pi)/omega
period.max <- round(max(period),2)
period.min <- round(min(period),2)

# plots the periodogram
plot(omega, IOmega, xlab="Angular frequency (rad) - [Top axis is period in
days]",ylab="I(omega)",bg="white",...)
axis(3,at=c(min(omega),0.5,1.0,1.5,2.0,2.5,3.0,max(omega)),
        labels=c(period.max,12.57,6.28,4.19,3.14,2.51,2.09,period.min))
title(main=title)
lines(omega,IOmega,type="h")

# displays periodogram
periodogram <- cbind.data.frame(period,omega,IOmega)
periodogram <- periodogram[order(periodogram$IOmega),]

retval <- periodogram[1:rows,]
return(retval)
}


envelope <-
```

```
function(object,resid.type="deviance",rep=20,epsilon=1e-3,maxit=1e2,plot=T,...)
# simulates and plots an envelope of residuals based on A. C. Atkinson book
{
st <- proc.time()

if (rep < 20)
        stop("Error: Number of replications must be greater than 20.")

dataset <- eval(parse(text=object$dataset))
n <- length(object$y)
fitted <- object$yhat[2:n]
formula <- as.formula(paste("SIMRESP",object$formula[1],object$formula[3]))
resid <- pgam.residuals(object,resid.type)[2:n]
e <- matrix(NA,n-1,rep)

attach(dataset)
for (i in 1:rep)
        {
        cat(paste("\nReplication:",i,"\n"))
SIMRESP <- rpois(n,fitted)
        runningmodel <-
pgam(formula,dataset=cbind.data.frame(SIMRESP,dataset),omega=model$omega,beta=ob
ject$beta,offset=object$offset,maxit=1e2,eps=1e-4,trace=F,optim.method="BFGS",pa
rtial.resid="response",numerical.se=F)
        # for now these will be the defaults ---> must be changed soon
runningresid <- pgam.residuals(runningmodel,resid.type)[2:n]
# for debugging
#cat(runningresid)
#cat("\n");cat(length(runningresid));cat("\n")
e[,i] <- sort(runningresid)
}

e1 <- numeric(n-1)
e2 <- numeric(n-1)

for (i in 1:(n-1))
        {
        eo <- sort(e[i,])
        e1[i] <- eo[round(0.025*rep,0)]
        e2[i] <- eo[round(0.975*rep,0)]
        }
residmean <- apply(e,1,mean)
band <- range(resid,e1,e2)

# plotting the envelope
if (plot)
{
qqnorm(e1,axes=F,main="",xlab="",ylab="",type="l",ylim=band,lty=1,lwd=1,col="re
d",bg="white")
par(new=T)
qqnorm(e2,axes=F,main="",xlab="",ylab="",type="l",ylim=band,lty=1,lwd=1,col="re
d",,bg="transparent")
par(new=T)
qqnorm(residmean,axes=F,main="",xlab="",ylab="",type="l",ylim=band,lty=2,col="b
lue",bg="transparent")
par(new=T)
qqnorm(resid,main="Simulated Envelope of Residuals", ylab="Residual
Component",xlab="Standard Normal Quantiles",ylim=band,bg="transparent",...)
}
et <- proc.time()
cat(paste("\nOverall envelope simulation process took",
elapsedtime(st,et)),"(hh:mm:ss)\n")

retval <- list(lb=e1,ub=e2,mean=residmean,residuals=resid)
class(retval) <- "envelope"
return(retval)
}


# functions end here -----------------------
```

B
CD-ROM com a Biblioteca pgam para R